$$\begin{tikzcd}
T_1 \arrow[dr, bend right] & \\
& T_2 \arrow[ul, bend right] \arrow[dl] \\
T_3 &
\end{tikzcd}$$

Fig. 11.5. Graph of precedences among transactions.

	T_1	T_2	T_3
		LOCK A	
		UNLOCK A	
			LOCK A
time			UNLOCK A
↓	LOCK B		
	UNLOCK B		
		LOCK B	
		UNLOCK B	

Fig. 11.6. A serializable schedule.

arc from T_2 to T_3. Steps (6) and (7) cause us to place an arc $T_1 \rightarrow T_2$. As there is a cycle, the schedule of Fig. 11.4 is not serializable. □

Example 11.7: In Fig. 11.6 we see a schedule for three transactions, and Fig. 11.7 shows its precedence graph. As there are no cycles, the schedule of Fig. 11.6 is serializable, and Algorithm 11.1 tells us that the serial order is T_1, T_2, T_3. It is interesting to note that in the serial order, T_1 precedes T_3, even though in Fig. 11.6, T_1 did not commence until T_3 had finished. □

Theorem 11.1: Algorithm 11.1 correctly determines if a schedule is serializable.

Proof: Suppose the precedence graph G has no cycles. Consider the sequence of transactions $T_{i_1}, T_{i_2}, \ldots, T_{i_r}$ that in the schedule S lock and unlock item A, in that order. Then in G there are arcs $T_{i_1} \rightarrow T_{i_2} \rightarrow \cdots \rightarrow T_{i_r}$, so the transactions must appear in this order in the constructed serial schedule. As no other transaction locks A, it is easy to check that the value of A after executing S is the same as in the serial schedule constructed by Algorithm 11.1. Since the above holds for any item A, it follows that S is equivalent to the constructed serial schedule, so S is serializable.

Conversely, suppose G has a cycle $T_{j_1} \rightarrow T_{j_2} \rightarrow \cdots \rightarrow T_{j_t} \rightarrow T_{j_1}$. Let there be a serial schedule R equivalent to S, and suppose that in R, T_{j_p} appears first

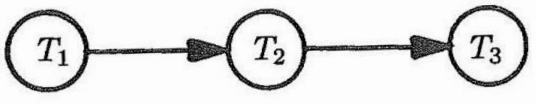

Fig. 11.7. Precedence graph for Fig. 11.6.

among the transactions in the cycle. Let the arc $T_{j_{p-1}} \rightarrow T_{j_p}$ (take j_{p-1} to be j_t if $p = 1$) be in G because of item A. Then in R, since T_{j_p} appears before $T_{j_{p-1}}$, the final formula for A applies a function f associated with some LOCK A—UNLOCK A pair in T_{j_p} before applying some function g associated with a LOCK A—UNLOCK A pair in $T_{j_{p-1}}$. In S, however, $T_{j_{p-1}}$ precedes T_{j_p}, since there is an arc $T_{j_{p-1}} \rightarrow T_{j_p}$. Therefore, in S, g is applied before f. Thus the final value of A differs in R and S, in the sense that the two formulas are not the same, and we conclude that R and S are not equivalent. Thus S is equavalent to no serial schedule. □

A Protocol that Guarantees Serializability

We shall give a simple protocol with the property that any collection of transactions obeying the protocol cannot have a legal, nonserializable schedule. Moreover, this protocol is, in a sense to be discussed subsequently, the best that can be formulated. The protocol is, simply, to require that in any transaction, all locks precede all unlocks.† Transactions obeying this protocol are said to be *two-phase;* the first phase is the locking phase and the second the unlocking phase. For example, in Fig. 11.3, T_1 and T_3 are two-phase; T_2 is not.

Theorem 11.2: If S is any schedule of two-phase transactions, then S is serializable.

Proof: Suppose not. Then by Theorem 11.1, the precedence graph G for S has a cycle, $T_{i_1} \rightarrow T_{i_2} \rightarrow \cdots \rightarrow T_{i_p} \rightarrow T_{i_1}$. Then some lock by T_{i_2} follows an unlock by T_{i_1}; some lock by T_{i_3} follows an unlock by T_{i_2}, and so on. Finally, some lock by T_{i_1} follows an unlock by T_{i_p}. Therefore, a lock of T_{i_1} follows an unlock of T_{i_1}, contradicting the assumption that T_{i_1} is two-phase. □

Another way to see why two-phase transactions must be serializable is to imagine that a two-phase transaction occurs instantaneously at the moment it obtains the last of its locks. Then the order in which the transactions reach this point must be a serial schedule equivalent to the given schedule. For if in the given schedule, transaction T_1 locks A before T_2 does, then T_1 surely obtains the last of its locks before T_2 does.

We mentioned that the two-phase protocol in is a sense the best that can be done. Precisely, what we can show is that if T_1 is any transaction that is not two phase, then there is some other transaction T_2 with which T_1 could be

† To avoid deadlock, the locks could be made according to a fixed linear order of the items. However, we do not deal with deadlock here, and some other method could also be used to avoid deadlock.

```
        .                 .
        .                 .
        .                 .
    LOCK A
        .
        .
        .
    UNLOCK A
                      LOCK A
                      LOCK B
                      UNLOCK A
                      UNLOCK B
    LOCK B
        .
        .
        .
    UNLOCK B
        .                 .
        .                 .
        .                 .

       T1                T2
```

Fig. 11.8. A nonserializable schedule.

run in a nonserializable schedule. Suppose T_1 is not two phase. Then there is some step UNLOCK A of T_1 that precedes a step LOCK B. Let T_2 be:

T_2: LOCK A; LOCK B; UNLOCK A; UNLOCK B

Then the schedule of Fig. 11.8 is easily seen to be nonserializable, since the treatment of A requires that T_1 precede T_2, while the treatment of B requires the opposite.

Note that there are particular collections of transactions, not all two-phase, that yield only serial schedules. We shall consider an important example of such a collection in Section 11.5. However, since it is normal not to know the set of all transactions that could ever be executed concurrently with a given transaction, we are usually forced to require all transactions to be two-phase.

11.3 A MODEL WITH READ- AND WRITE-LOCKS

In Section 11.2 we assumed that every time a transaction locked an item it changed that item. In practice, many times a transaction needs only to obtain the value of the item and is guaranteed not to change that value. If we distinguish between a read-only access and a read-write access, we can develop a